Formal description and evaluation of user-adapted interfaces

FIORELLA DE ROSIS AND SEBASTIANO PIZZUTILO
Dipartimento di Informatica, Università di Bari, Via Orabona 4, 70126 Bari, Italy. email: derosis|pizzutil @di.uniba.it

BERARDINA DE CAROLIS
Dipartimento di Informatica e Automazione, Università di Roma 3, Via della Vasca Navale 60, Roma, Italy. email: nadja@inf.uniroma3.it

(Received 23 January 1997 and accepted in revised form 11 March 1998)

This paper describes a visual formalism and a tool to support design and evaluation of human–computer interaction in context-customized systems. The formalism is called XDM (for “context-sensitive dialogue modelling”) and combines extended Petri nets with Card, Moran and Newell’s KLM operators theory to describe static and dynamic aspects of interaction in every context in which the system should operate, and to make evaluations of interface correctness and usability easier or automatic. The method was developed in the scope of a European Community Project to iteratively prototype a knowledge-based medical system. It has been subsequently employed in several research projects and in teaching activities.

1. Introduction

Properties and behaviour of increasingly complex user interfaces can hardly be described by informal methods, as this ambiguous specification increases the risk of misunderstandings with design partners and customers, and results in a long and expensive iterative revision of the prototype. A formally specified user interface contributes to translating user requirement analysis into the appropriate solution. More recently, formal methods have also been proposed for pre-empirical evaluation, before heuristic estimates and user testings are made, and to compare different interface alternatives (see for instance, Nielsen & Phillips, 1993, Byrne, Wood, Sukaviriya, Foley & Kieras, 1993).

Some people advocate that formal methods are difficult to use by people without expertise in the UI field or without mathematical basis; others claim that this risk is reduced when visual formalisms, tools or animations are employed. The question of which formal method to adopt is linked to the characteristics of the interface to be designed and to the specification method adopted. In this paper, we consider the case of multithread systems that are designed in close cooperation with representatives of users or customers. The formalism we propose is particularly suited to translate results of task analysis into the design of an interface that is customized to the context in which the system operates (Benyon, 1993). The formalism is called XDM (context-sensitive dialog modelling); it is supported by a tool that enables describing and simulating static and...
dynamic aspects of interaction, and evaluating the interface behaviour in the different
contexts in which the application is going to be used. We will describe XDM by drawing
examples from a European Union Project (OPADE), which initially promoted this
research. Section 2 introduces this project briefly and states why we selected Petri nets as
an underlying formalism for XDM and why we extended them. Section 3 describes XDM
in more detail and shows how it can be employed in a formal stepwise specification of
interfaces; the manner in which the interface behaviour in different contexts can be
simulated is described in Section 4. Section 5 specifies how the interface correctness can
be evaluated, and how some usability measures can either be made easier or performed
automatically by combining XDM with the KLM operators theory (Card, Moran
& Newell, 1980). We finally outline, in Section 6, how the method and its tool are going
to be validated and compare them with other methods in Section 7.

2. Why a multiple notation formalism

OPADE† was a European Community Project that involved several industrial and
academic partners to prototype and evaluate, in a three year term, a knowledge-based
medical system (De Rosis, Cozza, De Carolis, Errore, Pizzutilo & De Zegher, 1994).
Potential users of this system were health-care professionals with different levels and
types of experience, working in hospitals or in general practices. Re-implementation of
the interface consequent to iterative design had to be minimized in this project for
economic reasons. We therefore needed a formalism to specify static and dynamic
aspects of the dialogue in an “immediate” and “suggestive” way, to simulate the interface
behaviour in the different contexts in which the system was going to be used and to
evaluate intermediate versions in the iterative design process.

As a first step, we revised the visual formalisms and models that have been proposed to
represent a human–computer interaction. We noticed that each of them had its advant-
ages, and came to the conclusion that integration of different methods might be
convenient. In particular, we considered that graphical formalisms, which are extensions
of transition diagrams, might be more immediate and suggestive than others (such as
grammars), given that we had to cooperate with partners without a consolidated
experience in UI design. Among the various extensions of transition diagrams that have
been proposed, we selected Petri nets for several reasons. They enable us to describe
concurrent and asynchronous events which are typical of graphical interfaces and to
specify how the system state changes as a consequence of user actions (Van Biljon, 1988).
Coloured Petri nets (Jensen, 1983) open, in addition, the possibility of representing in
a unique structure, systems whose behaviours are varied according to the context, and
therefore seemed to us to be particularly suitable for formalizing user-adapted interfaces.

However, pure Petri nets do not enable the designer to fully describe all aspects of
interaction: how information about the system state appears in every phase of interaction
(interaction objects), which actions the user can perform (interaction techniques) and
how these physical aspects of the interface are related to information items and tasks to
be performed. We appreciated other authors’ proposals to enhance Petri nets with the

† “Optimization of drug Prescription using ADvancEd informatics”.
description of these “static” aspects of the interface in hypertexts (Stotts and Furuta, 1989): in Trellis, logical and physical “projections” of information displayed and user actions are associated with places and transitions. The same projections could be employed, in our view, to represent descriptions of system state and user actions in any interface.

The last requirement of our method was to automate (or at least to make easier) checking of the interface correctness and usability. Evaluations of some aspects of interface correctness is typical of the Petri net formalism. For example, as we will see in the following sections, the interface completeness is related to the reachability of the final state in the diagram. Others, like consistency, are enabled by our extension of Petri nets with logical and physical descriptions of the interface. To enable the evaluation of complexity, we integrated our formalism with KLM theory, in which the time required, on the average, to perform elementary and complex user actions can be estimated (Kieras, 1994).

In the following sections, after briefly introducing the basic PN concepts and terminology, we will describe how a user-adapted interface can be designed, simulated and evaluated with our formal method, by describing how all these functions can be performed with an XDM tool.

3. Stepwise formalization

XDM is a tool in C++ under Solaris 1, that includes several modules. It enables the designer to describe formally the interface, by a stepwise process, such as that listed below, which parallels the main phases of task-oriented design.

(a) Describe the dynamics of the dialogue by a Petri net.
(b) Nest the Petri net transitions, to reproduce task hierarchies.
(c) Describe static aspects of interaction (system states and user actions) by associating logical and physical projections to places and transitions.
(d) Add conditions to places and to transitions and colours to markings, to represent adaptivity to the context.

The mentioned phases are not strictly sequential; adaptivity is introduced, in particular, in the description of both the dynamic and the static aspects of the interface. The tool enables refining the interface design by iterative revision of the mentioned phases.

3.1. PRELIMINARY DEFINITIONS

Before describing XDM, we briefly introduce the Petri net concepts to benefit those who are not familiar with this formalism.

Petri nets (PN) are recursive transition diagrams with a “marking function” associated to places. The diagram’s states (places) are represented by circles (as we will see later on, we also represent some of them by stars and squares; we will describe the difference between these symbols). The transitions are represented by rectangles. A first example of PN is shown in Figure 1.

A PN can be formally defined as a tuple: \( \langle N, P, T, A, SP, EP, M \rangle \), where \( N \) is the name of the net; \( P \) is a finite set of places: \( p_1, p_2, \ldots, p_n \); \( T \) is a finite set of transitions:
FIGURE 1. An example of a Petri net.

$t_1, t_2, \ldots, t_m$; $A$ is a finite set of oriented arcs between $P$ and $T(T$ and $P);

\[ A : (P \times T) \cup (T \times P) \rightarrow \{0, 1\} \]

$SP$ is the set of start places (a subset of $P$); $EP$ is the end place (an element of $P$) and is represented by a square.

The set of places that are incident on a transition $t$ is a preset of $t(Pr(t))$:

\[ \forall p \text{ in } P: (p \text{ in } Pr(t)) \Leftrightarrow (A(p, t) = 1). \]

The set of places that follows $t$ is a postset of $t(Pr(t))$:

\[ \forall p \text{ in } P: (p \text{ in } Po(t)) \Leftrightarrow (A(t, p) = 1). \]

A path in the PN is a sequence of places and transitions that begins and ends with a place.

The function $M : P \rightarrow [0, 1]$ is called a marking function for the PN. This function maps each place $p$ to a binary value.

A place $p_i$ is marked if $M(p_i) \sim 1$ (a token is in that place), otherwise it is unmarked; marking is represented by a dot within the place. $M_0$ denotes the initial marking of the net, where all start places are marked. In the final marking $M_f$, the end place is marked. Markings propagate in the net according to the following rules: a transition is enabled when its preset is marked; it can then fire. After a transition has fired, its postset is marked and the markers in its preset are removed; the marking function represents how the system state evolves during execution.
In the example in Figure 1:

\[
P: \{ p_0, p_1, p_2, p_3, p_4 \},
\]

\[
T: \{ t_0, t_1, t_2, t_3, t_4 \},
\]

\[
A(p_0, t_0) = A(p_0, t_1) = A(t_0, p_1) \sim A(t_0, p_2) = \cdots = 1,
\]

SP: \{ p_0 \}, EP: p_4, \; M_4(p_0) = 1.

\(p_0\) is the preset of \(t_0\) and \(t_1\); \(p_1\) is the postset of \(t_0\) and the preset of \(t_2\); \(p_4\) is the postset of \(t_2\), \(t_3\), \(t_4\), \(t_5\), and so on.

Transitions \(t_0\) and \(t_1\) are enabled initially; when \(t_0\) fires, \(p_1\) and \(p_2\) are marked and \(p_0\) is unmarked: \(t_1\) is not enabled, now. The system is then “at state \(p_1\) and \(p_2\)”. Transitions \(t_2\) and \(t_3\) are now enabled; if \(t_2\) fires, the final marking is reached: \(M(p_4) \sim 1\).

### 3.2. DESCRIPTION OF MULTITHREAD DIALOGUES: PETRI NETS

When PNs are employed to describe the dynamics of a multithread dialogue, a PN represents a user task; for example, “Consultation of a patient record”. Places represent displayed information; transitions represent tasks performed by users, which cause a change in the display status.

A *path* in a PN is a sequence of places and transitions that begins and ends in a place. It represents the sequence of elementary tasks by which a more complex task can be performed and additionally shows how the display status changes during the execution of this task. In the example in Figure 1, \((p_0, t_0, p_1, t_2, p_4)\) is a path.

*Selection* between alternative paths is represented by associating the same preset to several transitions or, more in general, by enabling several transitions at the same time. For example, two alternative paths start from \(p_0\), in Figure 1; the first one is activated by \(t_0\), the second one by \(t_1\).

Alternative paths are closed by a unique place (\(p_4\), in our example).

*Visual concurrency* of information is represented by including several places into the postset of a transition or, more in general, by enabling several places to be marked at the same time: \((p_1, p_2)\) is an example of visual concurrency.

*Concurrently active paths*, which can be executed in any order, begin from these places. For example: \((p_1, t_2, p_4)\) and \((p_2, t_3, p_4)\).

*Synchronization* in the termination of these sequences is obtained by closing them with a single transition. In Figure 1, no synchronization condition is represented in the net.

For more details about PNs, see Van Biljon (1988), Hartson (1989) and Stott and Furuta (1989).

Following is a more complex example from OPADE. Figure 2 describes one of the tasks of this decision support system: the “Consultation of the Patient Record”. All examples, from this figure onwards, are displays generated by our tool. In XDM, Petri nets are built and updated with a non-commercial tool [Cabernet (Pezzeé, 1995)]. Places and transitions are named with capital letters in this tool. In the PN displayed on the left-hand side of window in this figure, the start place is \(p_0\): this place is the preset of \(t_0\) and \(t_1\), thus denoting that users can perform any of these tasks at the beginning of the interaction. The same holds for \(t_2\) and \(t_5\), from place \(p_1\), and for \(t_6\) and \(t_7\), from place \(p_4\). The two paths starting from transition \(t_2\) are, apparently, concurrently active; however,
as we shall see in the next section, they are activated in different contexts. The end place $p_8$, is the postset of transitions $t_4, t_5, t_{10}, t_{11}$ and $t_{12}$, which close the corresponding paths.

3.3. TOP-DOWN DESCRIPTION: NESTED PETRI NETS

In a nested Petri net, a labelling function $L$ is associated with transitions: this function maps each transition into the name of a PN or NIL: if $L(t) \neq \text{Nil}$, $t$ is a nested transition and is denoted by a white rectangle. We will name nested Petri nets “PN” like non-nested ones, as we do not need to distinguish between the two types of nets. Labelling transitions enable the designer to directly model hierarchical task analysis results: the dialogue is specified starting from the top-level task, and sub-tasks are described by calling a new sub-net. Each PN then corresponds to a task or a sub-task (Van Biljon, 1988). For example, in Figure 2, transitions $t_2, t_4, t_8, t_{10}$ are nested. If the designer clicks first on the “DESCRIBE” and then on the “CONDITIONS” buttons in the main bar, the two lists on the right side are displayed; one can look at the names of the lower-level PNs which are associated with nested transitions, in the bottom list: $L(t_2) = \text{Search for a patient}$, $L(t_4) = \text{Browse patient record}$ … and so on. This indicates that the “Consultation of the patient record” task can be decomposed into the following sub-tasks: Search for a patient, Browse through the patient record, Input new patient identification data and Create a new patient record. The first two of these tasks are to be performed in sequence;
others (“Create a new record” vs. “Browse through a record”) are performed alternatively. Some places in Figure 2 are represented by a star. These *macro places* correspond to a combination of elementary places whose meaning will become clearer in the next section, as well as the meaning of the asterisks associated with places and transitions.

3.4. STATE AND EVENT DESCRIPTION: DIALOGUE MODELS

The display status in each phase of interaction and the tasks that can be performed are specified in the following steps.

— In the first step *(logical level)*, the designer describes what the user is enabled to do and what the system does in response to the user’s stimuli: tasks that the user can perform in correspondence with each transition and information displayed in correspondence with each place.

— In the second step *(physical level)*, the designer defines how user tasks and system displays are implemented. This means specifying the user actions by which tasks are performed and the layout of displayed information.

A Dialogue Model is a tuple: \( DM = \langle PN, LP, PhP \rangle \), where \( PN \) is a Petri net and \( LP \) is its *logical projection*, defined as a couple \( \langle LP_p, LP_t \rangle \), where \( LP_p \) is a logical projection of places and \( LP_t \) is a logical projection of transitions.

Given a set \( G \) of tasks that the user wants to achieve and a set \( I \) of information items that the system will display as a result of these tasks; it follows that:

- The logical projection of places \( LP_p \) is a function that associates to each place of the PN a description of information displayed: \( LP_p = P \rightarrow I \).
- The logical projection of transitions \( LP_t \) is a function that associates to each transition of the PN a description of the task performed: \( LP_t = T \rightarrow G \).

Figure 3 shows an example of logical projection of places and transitions for the PN in Figure 2. To display these lists, the designer clicks on the “LOGICAL PROJ” button in the main bar, after having selected a “context”, according to criteria that will be specified in the next paragraph. The two lists on the right-hand side describe, respectively, the information that is displayed when each place becomes marked and the task which is associated with each transition. The interaction starts from the display of the “Clinical record selection command” \( p_0 \). The user can perform two, alternative, tasks, “Access to an existing patient record” \( t_0 \) or “Creation of a new patient record” \( t_1 \). In the first case, the system will display the “Main patient identification data” that will have to be specified to select the patient, with a “confirm” command \( p_1 \). There are two alternatives now: just “Exit from the task” \( t_5 \) or go on with the “Search for a patient” task, by filling the main identification data form \( t_2 \). The designer may ask to look at the subnet associated to \( t_2 \) by clicking on the “DOWN” button in the main bar, or may continue analysis of the main PN. Let us consider this second case: the main PN shows that, after performing the sub-net associated with \( t_2 \), the two macroplaces in its postset will be marked \( p_2 \) and \( p_4 \). We will see in the next section that this is, in fact, an apparent concurrent display of items, for reasons that we will clarify later on.

Let us now consider, in particular, the place \( p_4 \). When this place becomes marked, a warning message “Patient not found” is displayed. The user can either exit from this
display to return to \( p_i \) (through \( t_i \)) or ask to create a new patient record, through \( t_7 \). The identification data of the new patient record will be displayed when \( p_5 \) becomes marked and, if the user confirms them (\( t_{11} \)), the end place \( p_8 \) is reached and this task is completed. In the same way, and by firing different transitions in the postset of places (for instance, \( t_0 \) rather than \( t_1 \) initially), the designer can examine all paths which lead to the end place, to get an overall idea of the interface behaviour in the "Consultation of the patient record" main task.

We remark that, until this phase of interface modelling, no specification of the physical aspects of the interface has yet been made. The specification can be made by attaching to the PN a description of the physical properties of places and transitions, \( \text{PhP} \).

The physical projection \( \text{PhP} \) of a PN is a couple \( \langle \text{PhP}_p, \text{PhP}_t \rangle \), where \( \text{PhP}_p \) is the physical projection of places and \( \text{PhP}_t \) is the physical projection of transitions.

Given a set \( A \) of actions that the user can perform and a set \( D \) of screen layouts that the system can show as a result of these actions as follows:

- The physical projection of places \( \text{PhP}_p \) is a function that associates to each information item in \( \text{LP}_p \) its screen layout \( D: \text{PhP}_p = I \rightarrow D \).
- The physical projection of transitions \( \text{PhP}_t \) is a function that associates to each task in \( \text{LP}_t \) the action that the user has to perform to achieve it: \( \text{PhP}_t = G \rightarrow A \).

Each element of the set \( D \) is implemented by one of the available interaction objects \( C_I \). Each element of the set \( A \) is implemented by one of the available interaction techniques \( C_A \). \( C_I \) and \( C_A \) are functions of the environment in which the system is developed.

For example, the environment employed in developing OPADE was XViewLib, which provides the elements shown in Table 1. Interaction objects can be assembled into
Table 1
Interaction objects and techniques in XViewLib

<table>
<thead>
<tr>
<th>Interaction objects</th>
<th>Interaction techniques</th>
</tr>
</thead>
<tbody>
<tr>
<td>Settings (text or icon):</td>
<td>$S_S = $ Select a setting item</td>
</tr>
<tr>
<td>$S_{ti}^i$ = exclusive</td>
<td></td>
</tr>
<tr>
<td>$S_{ti}^{ii}$ = stack</td>
<td></td>
</tr>
<tr>
<td>$S_{bi}^i$ = check box</td>
<td></td>
</tr>
<tr>
<td>Buttons:</td>
<td>$S_B$, or $S_{bi} = $ Select a button or an icon:</td>
</tr>
<tr>
<td>$B_i = $ text</td>
<td></td>
</tr>
<tr>
<td>$B_i = $ icon</td>
<td></td>
</tr>
<tr>
<td>Messages</td>
<td>$S_M = $ Select an item from a menu</td>
</tr>
<tr>
<td>$C_i = $ text</td>
<td></td>
</tr>
<tr>
<td>$C_i = $ icon</td>
<td></td>
</tr>
<tr>
<td>$M = $ menus</td>
<td>$F_T(n) = $ Fill a textfield of $n$ characters by typing a string</td>
</tr>
<tr>
<td>$T = $ Textfields</td>
<td>$S_{L}(v, w) = $ Select an item from a scrolling list of $v$ items, $w$ of which are displayed at the same time</td>
</tr>
<tr>
<td>$L = $ Scrolling lists</td>
<td></td>
</tr>
</tbody>
</table>

Macro interaction objects. For example: $GB_i$, $GB_i$ are groups of icon or text buttons; $GT$ is a group of textfields; $MB_i$, $MB_i$ are menus associated with an icon or a text button. We include windows among the macro interaction objects as they are a combination of elementary interaction objects.

Macro interaction techniques can be defined as operations on macro interaction objects. For example, filling a window with $r$ textfields of $n$ characters is the repetition ($r$ times) of the action of filling a textfield of $n$ characters: $F_{W(n)} \sim (F_{T(n)})^r$.

After building the PN with Cabernet, the designer specifies the logical aspects of the interface by associating a logical projection table to places and to transitions. To fully describe its physical layout, all images appearing in different contexts and phases of the dialogue have to be generated, either on paper or by means of an interface builder. Images shown in this paper were made with DevGuide, as well as XDM’s interface. The designer can then link the database of these GUI elements to the Petri net by means of the PhP function.

Let us look at Figure 4, which shows part of the physical projection of our PN. The designer can examine the physical projection of a transition or a place in a specified context by clicking first on the “CONTEXT” icon and then on the “PHYSICAL PROJ” button in the main bar, and by finally selecting a transition or a place. In this figure, the two projections of $t_0$ show that the “Access to an existing patient record” task is performed, in two different contexts (when the user is a physician (left-hand side) and when he/she is a nurse (right-hand side)) by selecting two different icons. The window of the bottom shows how the “Patient identification data” associated with the macro place $p_2$ are displayed in the indicated context.

The combination of a nested PN and of its logical and physical projections then provides a complete description of static and dynamic aspects of interaction, for a
specific system whose interaction style is defined by the environment employed to develop it. The logical projection is still device-independent, whereas the physical projection is not.

3.5. MULTIPLE VIEWS IN A SINGLE MODEL: CONTEXT-SENSITIVE DIALOGUE MODELS

We mentioned several times the concept of "context" in our interface description. Context-dependent description of interfaces is needed when user requirement analysis shows that the system will be employed in several contexts with different needs. In this case the designer may decide to implement an interface that is able to behave differently in the different contexts. A context can be a combination of several features: the site in which the system is installed (in the case of OPADE, hospital, general practice or others), the country, the user’s job (physician, nurse, health administrator, etc.) or other user characteristics. The formal model must describe, in this case, how interaction varies with the context. XDM provides this opportunity, as it unifies in a single model a complete description of how the interface will behave in all contexts for which the system has been designed; the way that access rights to specific tasks are restricted to specific contexts and the way in which tasks are performed and information is displayed in each context. Let us examine these adaptation features in more detail.

3.5.1. Adaptation of dynamic aspects of the dialogue

User adapted dialogues can be represented by “colouring” the Marking Function of the PN, that is by changing its definition as follows: $M : P \rightarrow \Phi$, where $\Phi$ is a set of "token
colours.” A token colour is a non-quantified formula in a first-order language \( \Sigma \). Differently coloured tokens denote distinct contexts in which the system might be employed.†

For example, consider the following:

- A particular category of OPADE users, hospital doctors, can be described as:
  
  \((\text{UserGroup(PHYS) AND Site(HOSPITAL)})\).

- A particular category of patients (inhospital male patients) can be described as:
  
  \((\text{AdmissionMode(INPATIENT) AND PatientSex(MALE)})\).

Colours can be employed to specify, as well, the state of the application; for instance, PatientRecord(FOUND) means that the procedure of “Searching for a patient” with specified identification data was successful.

A context-sensitive dialogue is described by a coloured nested Petri net, which is a tuple \( \text{CPN} < \langle \text{PN}, F, G \rangle \) where \( F \) is a place-conditioning function that maps each place \( p \) into a nonquantified ground formula \( \text{Cond}(p) \) in \( \Sigma \) or \( \text{Nil} \); this formula specifies a condition for the place to be marked; and \( G \) is a transition guard function that maps each transition \( t \) into a non-quantified ground formula \( \text{Tg}(t) \) in \( \Sigma \) or \( \text{Nil} \); this formula specifies a condition for the transition to fire.

Figure 2 shows that the \( F \) and \( G \) functions for the “Consultation of the Patient record” task. Starred transitions and places have a condition associated with them, which are shown in the two pop-up windows on the right-hand side of the figure.

\( F(p_2) = \text{PatientRecord(FOUND)} \) and \( F(p_4) = \text{NOT PatientRecord(FOUND)} \) associate mutually exclusive conditions to places \( p_2 \) and \( p_4 \); place \( p_2 \) can be marked only if the search of the patient record was successful, whereas, if the record was not found, place \( p_4 \) can be marked instead.

\( G(t_1) = \text{UserGroup(PHYS)} \) and \( G(t_7) = \text{UserGroup(PHYS)} \) associate the same condition to transitions \( t_1 \) and \( t_7 \); these transitions are enabled only when the user is a physician.

**Coloured marking propagation rules** are defined as a function of \( F \) and \( G \). A transition \( t \) is enabled in a context that depends on its transition guard and on the marking of its preset \( M(\text{Pr}(t)) \). This marking is the conjunction of markings of places in the preset. If \( \text{Tg}(t) \) is associated to \( t \), then \( t \) will be enabled in the context: \( E(t) = M(\text{Pr}(t)) \AND \text{Tg}(t) \).

Let \( p_j \) be an element of the postset of \( t \) which has no other incoming transitions, and \( \text{Cond}(p_j) \) the condition associated to \( p_j \). After firing \( t \), the new marking of \( p_j \) will be

\[
M_{\circ}(p_j) = E(t) \AND \text{Cond}(p_j).
\]

If \( p_j \) is the postset of several transitions \( t_1, t_2, \ldots, t_h, \ldots, t_q \), each being enabled in a context \( E(t_h) \), the marking of \( p_j \) will be updated progressively, each time one of its antecedent transitions \( t_h \) fires, as follows:

\[
M_{\circ}(p_j) = M(p_j) \AND E(t_h) \AND \text{Cond}(p_j).
\]

† From now on, we will denote (AND, OR, NOT) the logical connectives by capital letters, and their natural language homologues by lower case letters.
For each $p_j$ in the preset of $t$, after firing $t$, the new marking $M^o(p_j)$ will become as follows:

$$M^o(p_j) = M(p_j) \text{ AND Cond}(p_j) \text{ AND NOT } E(t).$$

When a nested transition $t$ fires, the start places in the associated subnet are marked with the same colours for which the transition was enabled. When the end place of the subnet is reached, the postset of $t$ is marked with the same colours as the end place of the subnet. The previous rules then define how markings propagate in the situations of sequence, visual concurrency and selection.

Let us look again at Figure 2, and let us set the following initial marking: User-Group(NURSE). Only the transition $t_0$ is enabled in the defined context; when it fires, $p_1$ takes the initial marking and $t_2$ can fire in the same context. The two elements of the postset of $t_2$ ($p_2$ and $p_4$) are now marked differently:

(UserGroup(NURSE) AND PatientRecord(FOUND)) for $p_2$ and

(UserGroup(NURSE) AND NOT PatientRecord(FOUND)) for $p_4$.

Transition $t_1$ is not enabled in this context, so only $t_3$ or $t_6$ can fire; $p_3$ is marked with (UserGroup(NURSE) AND PatientRecord(FOUND)), $t_4$ can fire in the same context and $p_8$ is finally marked in the same way.

This marking propagation example shows that nurses can perform the “Consultation of the patient record” task only through the examined path, that is by first searching for a patient and then browsing through the corresponding record: they cannot, on the contrary, create a new record by going through any path starting from $t_1$ or passing through $t_7$.

3.5.2. Adaptation of state and event descriptions

Logical and physical layouts can be adapted by introducing colour constraints into LP and PHP functions. A context-sensitive dialogue model is a tuple:

$$\text{XDM} = \langle \text{CPN}, \text{CLP}, \text{CPhP} \rangle,$$

where $\text{CPN}$ is a coloured, nested Petri net; $\text{CLP}$ is a coloured logical projection of places and transitions; $\text{CPhP}$ is a coloured physical projection of places and transitions; $\text{CLP} = \langle \text{CLP}_p, \text{CLP}_t \rangle$, where $\text{CLP}_p, \text{CLP}_t$ are the coloured correspondents of the logical projections of places and transitions $\text{LP}_p, \text{LP}_t$ that we described in Section 3.4. Introducing colours just means introducing an additional condition on the context $\Sigma$, in the left-hand of these projections:

$$\text{CLP}_p = P \times \Sigma \rightarrow I, \quad \text{CLP}_t = T \times \Sigma \rightarrow G,$$

The same holds for the coloured physical projections of places and transitions:

$$\text{CPhP} = \langle \text{CPhP}_p, \text{CPhP}_t \rangle,$$

with

$$\text{CPhP}_p = I \times \Sigma \rightarrow D, \quad \text{CPhP}_t = G \times \Sigma \rightarrow A.$$

Single elements of the logical and physical projections of places and transitions describe the information that is displayed when a place is marked and the task that is performed when a transition is fired in a specific context. For example, consider again Figure 4. The two top windows show the two physical projections of transition $t_0$ in the contexts
UserGroup(PHYS) and UserGroup(NURSE). In this example, two different icons are associated to the same task in the two contexts. This is one of the results of an experimental study about the iconic language of OPADE (De Carolis, Errore, De Rosis, 1995), which showed that the two icons were considered as particularly suggestive by the two types of users, probably because they evoke the different ways in which physicians and nurses perform the task of looking at a patient's record.

Another example of adaptation of the physical layout is given by the two patient identification windows in Figures 4 and 5. The first one is displayed in the context “AdmissionMode(INPATIENT)”, the second one in the context “AdmissionMode(OUTPATIENT)”, both when Site(HOSPITAL) is true. In the first window, data about the way the patient was admitted, the reason for intake, the unit, ward and bed are specified; in the second one, the patient’s address is specified instead. Other differences are due to the difference of the patient’s sex.

By means of the Coloured logical and physical projection functions, one can describe, more in general, how database management tasks are customized to the user’s level of experience (for instance, an iconic language for naive users and a command language for more experienced ones) or how the iconic language is adapted to the user’s cultural background. A complete logical description of user tasks performed in all contexts in which the system will operate is obtained by associating, with every transition, a two-way table of tasks performed, one dimension of this table being the context description. The same can be done for the logical description of information associated to places.

Two-way tables have to be also associated to the physical descriptions of tasks performed, and of information displayed, for a complete description of the interface appearance in all contexts.

4. Simulation

After describing the static and dynamic aspects of the interface, the designer can simulate the system behaviour in a specific context, by clicking on the “SIMULATE” button in the menu bar. The marking function is set to the initial marking colour that corresponds to that context, and only one token with that colour is placed in all start places. As mentioned in Section 3.1, a transition is enabled when all places in its preset are marked. When a transition is enabled, a pop-up message window indicates the action that the user can perform and the task that is achieved with this action. When a transition fires, the tokens exit from its preset and enter into its post set. When a token enters into a place, the physical projection of that place is displayed. When a token exits from a place, that information is undisplayed. Firing of a transition thus results in a change of information displayed. A particular case of change occurs in data entry tasks, where an empty field or window is substituted with a filled field or window. The simulation halts when either no transition is enabled (unsuccess) or the end place is marked (success).

The interface behaviour in a particular context can be simulated in an “AUTOMATIC” or a “STEPWISE” way, by clicking on the corresponding buttons in the menu bar. In the first case, when the system comes to a situation in which several transitions are enabled, it asks the designer to select one of them. In the stepwise mode (which was selected in the example in Figure 5), the system stops at every transition, requiring the designer’s intervention. When the dialogue comes to a nested transition, the designer can
either “go down” to simulate the corresponding task (by clicking on the “DOWN” button), or go on with the simulation in the same, top-level PN. Figure 5 shows the display status at an intermediate stage of simulation. The list on the top right-hand side shows the history of simulation steps ($t_0$ fired, $p_1$ displayed, $t_2$ fired and $p_2$ displayed). The window below it shows the physical projection of place $p_2$. A pop-up message asks the user to fire $t_3$ by pressing the “YES” button and sends a reminder that the task associated with this action is “Confirm the Patient Record Selection.”

5. Formal evaluation

Let us adopt a common distinction between two categories of evaluation: (1) correctness establishes whether the interface enables its users to perform correctly all tasks that they should perform and only those; (2) usability is “the degree to which specified users can achieve specified goals in specified environments, subject to various comfort, effectiveness and acceptability criteria” (citation in Thimbleby, 1994). We now show how our formalism supports both types of evaluation.

5.1. CORRECTNESS

According to the above definition, an interface is correct when it is complete and non-redundant and when access rights are respected, so that users are enabled to perform only the tasks they are entitled to.
FIGURE 6. UI evaluation: an example of control of completeness and respect of access rights.

5.1.1. Completeness

Completeness in a context is a synonymous reachability of the end place of the CPNs that represent the tasks that should be performed in that context. A task can be performed in a specified context if and only if given an initial marking whose value corresponds to that context, its end place is marked with the same value. Reachability can therefore be calculated automatically, by applying the coloured marking propagation rules that were described in Section 3.5. Figure 6 shows an example of verification of the interface completeness, for the sub-task “Search for a patient” which was associated with transition $t_2$ in the main net. The context in which the designer checks for completeness is indicated at the top of the list: (UserGroup(PHYS) AND Site(HOSPITAL)).

This text view shows that this task can be performed in several ways in the selected context. We do not show logical and physical projections of this sub-net, for space reasons. We will only say that places $p_{1.1}, p_{1.2}$ and $p_{1.3}$ are the expansion of the macroplace $p_1$ in the main net ConsPR. They are associated (respectively) with the record identification, the last name and the list of patients. The three paths in this sub-net correspond to different ways of selecting a patient; by specifying the record identification (path 1) or the last and the first name (path 2), or by selecting the patient from a scrolling list (path 3).

An example of the verification of unreachability in the same figure is as follows: patients in a general practice cannot be selected according to their record identification as this code is only defined for hospital inpatients. This is ensured by attaching, to $p_{1.1}$, the condition: $\text{Cond}(p_{1.1}) = \text{Site(HOSPITAL)}$. In XDM, when the end place is not reachable through a path, the designer can ask the system to verify the cause of the deadlock (in our example, the condition associated to $p_{1.1}$).
5.1.2. Respect of access constraints

One has not only to check that all tasks that should be performed in a context are, in fact, enabled by the interface, but also that access rights to tasks are not violated. Access rights set up, for example, bindings on the tasks that each category of users is entitled to perform. In Section 3.5.1, we gave an example of how to represent the condition that nurses are not entitled to create new patient records. Checking for access constraints requires making sure that the end place of a CPN cannot be reached through a path that corresponds to the forbidden task, when the initial marking of the CPN is settled to the colour that corresponds to the context in which the task is forbidden.

5.1.3. Non-ambiguity

A human–computer interface is said to be “redundant” when different user commands lead to the same display status: redundancy thus corresponds to alternative ways of doing a task, and should not be considered a negative issue. For example, in Figure 6. Search for a Patient could be done in different ways by hospital doctors: in this case, the end place of the alternative paths was unique, and represented completion of the task. Irrespectively of whether an interface is redundant or not, it should not be ambiguous. Non-ambiguity equates to the Dix and Runciman’s Program-Interpretation-Effect correspondence (Thimbleby, 1990), which can be specified, in our view, in the following two conditions.

— The user must be sure that, when he/she performs a command (or a sequence of commands), this will always bring the system to the same state.
— By looking at the display status, the user should always understand which the system state is (the “gone for a cup of tea” problem).

If we agree on the statement that the system state corresponds to the degree of performance of a given task, the two non-ambiguity conditions can be translated into the following checks.

- The same sequence of user commands should always perform the same task in the same context, and lead to the same display status (Hsia, Samuel, Gao, Kung, Toyoshima & Chen, 1994).
- The same display status should, inversely, correspond to the same level of performance of a given task in a given context.

In this sense, one of the origins of ambiguity of interfaces is non-determinism of finite automata that describe them; in XDM, determinism in a Petri net is translated into the constraint that different transitions starting from the same place denote distinct tasks and therefore correspond to distinct CLP, table items. If a complex task is described in a unique Petri net, this check can be performed automatically. This criterion for avoiding ambiguities parallels the suggestion to avoid repetitions of the same node, in Hierarchical Task Analysis (Macawley, 1995). A more general check of the first type of ambiguities requires examining the physical projections of all transitions to verify that, in a given context, it does not happen that equal user commands are associated with different tasks.

As for the second type of checks, physical projection of places has, again, to be examined, to verify that it does not happen that the same visual display content is associated with different information items. In XDM, information items are also
employed to make the application state explicit. This requires verifying, in particular, that distinct places with the same physical projection do not denote different stages of completion of a given task, in distinct phases of interaction. For instance, a display status that denotes the successful completion of a task performed by a given sequence of commands should not denote, in another phase of interaction, partial or erroneous performance of the same task, either when this is made with the same or with a different sequence of commands.

5.2. USABILITY

Several empirical criteria have been proposed to evaluate the usability of interfaces. For example, the time to learn or to perform a task or the rate of errors made in learning or in performing it. The relevance of these parameters depends on the expected use of the system (discretionary vs. compulsory, intermittent vs. regular) and on the duration of the expected user-to-system interaction (short vs. long) (Badre, Chalmers, Copeland, Levi-ldi, Mussio & Solomon, 1994).

The way in which empirical usability measures relate to design features that can be evaluated formally has not yet been defined clearly. Thimbleby proposes to measure usability in terms of interface complexity in different conditions of user knowledge. His measures are based on counts of user actions that are needed to perform a task or on estimates of the length of complete user manuals. The first parameter is, to the author, a function of the structure of the finite-state machine that represents the interface; the second one depends on the tree structure of the manual itself (Thimbleby, 1994). Interface complexity is directly estimated, in UIDE, from physical actions using NLGOMS (Byrne et al., 1994). On the other hand, a wide variety of studies have shown that time and errors in performing a task increase as the amount of information in the display increases and that, consequently, at all points in the user’s interaction with the system only information needed to perform the task should be displayed (Tullis, 1988). Display complexity is also the metrics that Kim and Foley apply automatically to analyse visual properties in their knowledge-based design of interfaces (Kim and Foley, 1993).

Thimbleby suggests that consistency criteria should also influence usability, without specifying how this could be measured formally (Thimbleby, 1994). Although the meaning of this term remains elusive, the role of consistency in usability has been established “in negative,” after “a number of researchers have shown, in behavioural experiments, that inconsistencies of various kinds result in systems that are harder to use than their consistent counterparts” (Reisner, 1993). More specifically, Tero and Briggs (1994) showed a strong association between action consistency and ease of learning, in an experimental study in which consistency was measured by the degree of correspondence between some simple tasks and their iconic representations, and Phyllis Reisner reasoned about user errors that would occur as a result of inconsistency of actions (Reisner, 1993). To all authors, interface consistency is linked to “doing similar things in similar ways”. However, several studies show that the ways in which data are displayed (grouping of elements and their placements, sequence, presentation form and so on) play an important role in both the ease in which the users can extract the information and the interpretations that they assign to it (Tullis, 1988). For this reason, we propose to extend the previous sentence by saying that interface consistency is also linked to “seeing similar
"things in similar ways" and that consistency depends, like complexity, on the way in which user actions are linked to tasks and information items are displayed: a property which is studied, for instance, through the task-to-device mapping in GOMS or through the definition of semantically similar sets of elements in APT (Reisner, 1993).

Based on the analysis of previous studies, we assumed that complexity and consistency are the two main features that should be estimated in our formal models, and that each of them should be examined in terms of user actions needed to perform tasks and display status after these actions have been performed. We will describe, in the next two sections, how XDM supports automatic or semiautomatic evaluation of these features.

5.2.1. Consistency

If the interface is not consistent, the field user will have difficulty in associating the external behaviour of the system with its internal functioning. This concept has been the object of a long debate among those involved in HCI research (see, for instance, Grudin, 1989; Reisner, 1993; Tero & Briggs, 1994). According to this debate, analysis of consistency has to be extended to consider ‘similar’—and not only ‘identical’—user tasks and information items displayed. In XDM, actions that a user has to make to achieve specific tasks in a specific context are consistent across the whole dialogue if similar tasks that are attached to different transitions are performed, by that category of users, by similar actions (action consistency). For example, (1) saving a record in different phases of the dialogue is always made by clicking on the same type of button and (2) icon commands to perform different database management functions employ the same symbol to denote the database (De Carolis et al., 1995). Likewise, the display layout is consistent if similar information items are displayed, in different phases of the dialogue, by similar layouts and if, when displayed at the same time, they are assembled into the same window or are located in the same region of the screen (display consistency).

Analysis of action and display consistency in each context cannot be made automatically in our formalism. This would require representing the semantics of each item (elementary task and information) in an application-specific domain knowledge base. This representation is not easy, also considering that internal consistency (similarity across the system) can be different from external consistency (similarity between the user's and the designer's views). However, XDM can assist the designer in this analysis by exploiting knowledge in logical and physical projection tables. In (internal) action consistency analysis, all entries of the logical projection tables CLP, and CPhP are sorted according to the description G of the task performed. These descriptions are then tabulated by displaying the name of the PN in which the task is performed, the name of the transition to which the task is attached, the interaction technique adopted to perform the task and its physical layout. The designer can check whether action consistency is respected by comparing descriptions of identical or similar tasks in different PNs.

Figure 7 shows an example of this analysis. The designer clicks on the “CONSISTENCY” button and selects “Action” from the pull-down menu. The displayed table shows that the “Access to an existing Patient Record” task can be performed in two different phases of the interaction: in the Petri net ConsPR (which is shown in Figure 2), by means of transition $t_0$; in another net which is not shown in this paper (PrescrPR), by means of transition $t_{12}$. The table shows that this task is not performed consistently in the two phases. In the first case, it requires a “Select icon” interaction technique ($S_{Bi}$),...
followed by a ‘Select a menu item’ one ($S_M$). In the second case, it requires a ‘Select a text button’ ($S_{BT}$), followed by a ‘Select a menu item’ ($S_M$).

The task of ‘Creation of a new patient record’, which follows immediately in the same table, is similar to the previous one as it applies a different process (creation rather than access) to the same object (the patient record). The table shows that the two similar tasks can be performed by similar interaction techniques in the PN ConsPR. That is “Select icon” + “Select a menu item”. They are, therefore, consistent. In addition, by double clicking on a row in the table, the designer can check whether the physical projections of these actions are consistent. In our example, this icon is consistent with the icon employed for Accessing an existing Patient Record, which we saw in Figure 4. Display consistency can be examined in a similar way, by checking that the same information items are displayed in the same way across the interface. That is, that each of them has a uniform presentation (interaction object used, labels, coding criteria and so on).

A much more complex procedure would be required to evaluate the consistency of action sequences needed to perform complex tasks, and of screen layouts in which multiple data are displayed at the same time. Comparison of action sequences corresponding to similar tasks would require the analysis of a similarity of strings in the alphabet of interaction techniques. The comparison of screen layouts that display similar macro-information items (such as “Patient identification data”) in two different phases of the application would require verifying that these items are presented in the same order, assembled in a similar way on the screen and so on.

### 5.2.2. Complexity: integration with KLM operators

The complexity of an interface has been defined as “the amount, content and structure of the knowledge required to operate the device successfully” (Kieras and Polson, 1985). We decompose, again, this measure into an action complexity and a display complexity.

The complexity of the sequence of actions needed to perform a task in a specific context can be measured as a function of the length of the string that represents the sequence of transitions that lead to the end place when the initial marking of the CPN representing that task is set up to the colour that corresponds to the context. If several,
Table 2

Interaction techniques in XViewLib, as a function of KLM standard primitive external operators

<table>
<thead>
<tr>
<th>Interaction Technique</th>
<th>Symbol</th>
<th>KLM expression</th>
<th>Time units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select a button</td>
<td>$S_B$</td>
<td>pbb</td>
<td>13</td>
</tr>
<tr>
<td>Select a setting or a menu item</td>
<td>$S_M$, $S_S$</td>
<td>pbpp</td>
<td>24</td>
</tr>
<tr>
<td>Select an item from a scrolling list of $v$ elements, $w$ of which are displayed at the same time</td>
<td>$S_L(v, w)$</td>
<td>$\epsilon + pbb^q b + (pbb)pbb$</td>
<td>$(26 + 11q)$ or $(13q + 1)$</td>
</tr>
<tr>
<td>Fill a textfield of $n$ characters</td>
<td>$F_T(n)$</td>
<td>pbbhk$^n$</td>
<td>$17 + 2n$</td>
</tr>
<tr>
<td>Fill a window with $r$ objects, each being a setting or a textfield of $n$ characters</td>
<td>$F_W(n</td>
<td>r)$</td>
<td>$(pbb(hk^n + p))^r$</td>
</tr>
</tbody>
</table>

alternative paths enable reaching the end place in a given context, one can compare their complexities. One can compare, as well, the complexities of actions needed to perform the same task in different contexts; for instance, by different user profiles. This comparison can be employed to check that tasks are easier to perform for less expert users or in situations in which interaction speed is essential.

However, elementary actions employed to perform elementary tasks may have different complexities; selecting a button is less complex than filling a textfield of $n$ characters. We measure the complexity of elementary interaction techniques by expressing them as a function of KLM standard primitive external operators (Kieras, 1994): press or release a mouse button (b), move the cursor to point to a target coordinate (p), press a key (k), homing (h). A XDM interaction technique can therefore be described as a regular expression in the language of KLM operators, by denoting, as usual, the union with $\cup$, the $n$-times iteration with an exponential and the empty string with $\epsilon$ (see Table 2). The complexity of an interaction technique can be estimated as a function of its expected execution time, from the KLM operators expression. Kieras (1994) estimates execution times to be (1) 0.1 and 1.1 s, respectively, for b and p operators, (2) 0.2 s, on the average, for a skilled typist for k and (3) 0.4 s for h. The complexity of XDM interaction techniques can therefore be expressed in units of tenths of seconds, as shown in Table 2.

In the KLM expression for $S_L(v, w)$, $q = v - w$ is the number of “hidden” items. If the item to be selected is one of the $w$ elements displayed initially, one has to locate it and press and release the mouse button (pbb), otherwise the list must first be scrolled by keeping the mouse button pressed (pbb$^q$b) or by clicking $q$ times on the button (pbb$q$), until the item appears. In performing $F_T(n)$, one has to position the pointing on the textfield and type the string. The time units needed to perform $F_W(n|r)$ are (pbb$^q$) when

† Selecting an item from a scrolling list of $v$ items, $w$ of which are displayed at the same time.
‡ Fill a textfield of $n$ characters.
§ Fill a window with $r$ objects, each being a setting or a textfield of $n$ characters.
all inputs are setting selections, \( (\text{pbbhk}^n) \) when they are all textfield fillings, an intermediate value for mixed interaction techniques.

Figure 8 shows an example of how this method can be applied to compare the complexity of alternative ways of performing a task. As we have seen in Figure 6, the “Search for a patient” task can be performed by hospital doctors, that is in the context \((\text{UserGroup(PHYS) AND Site(HOSPITAL))}\), in three alternative ways.

(a) By specifying the record identification (path 1): \( \text{Compl}(F_T(n)) = 17 + 2n \) units.

(b) By specifying the last and the first names (path 2): same interaction techniques and same complexity, but as a function of the length of two names: \( \text{Compl}(F_T(m_1) + F_T(m_2)) = 34 + 2m_1 + 2m_2 \) units.

(c) By selecting a patient from the scrolling list (path 3): complexity ranging from 13 to \((26 + 11q)\) or 13\((q + 1)\) units.

If we make some assumption about the parameter values (for example, \( n = 4, m_1 = 8, m_2 = 7, q = 90 \)), we can estimate the complexity of the three alternatives (25, 64 and from 15 to 1183 units, respectively). By clicking on the button “COMPLEXITY” and selecting the pull-down menu item “Action”, the designer will obtain these measures and will have the possibility of evaluating the relative advantage of using each alternative. It should be noticed that these estimates of complexity omit “mental operations” such as “locate an object on the screen” or system answer times, that are considered in GOMS operators. They are therefore purely manual operation times.
The display complexity is a function of the amount of information that is displayed at the same time. For example, the number of items in simultaneously open windows. Display complexity in a specific context and in a given phase of interaction is a function of the number of places that are marked in the CPN in that context and in that phase of interaction, and of the complexity of each macroplace.

Let us consider, for example, the CPN in Figure 4 and let us assume that we are in a phase of interaction in which only the macroplace $p_2$ is marked. The complexity of $p_2$, in the context specified in that figure:

(UserGroup(NURSE) AND Site(HOSPITAL) AND AdmissionMode(INPATIENT) AND PatientSex(M))

is equal to $3B_i + 1L + 4S_i + 10T$ (three buttons, one list, four stack text settings and 10 textfields). If we compare this figure with the complexity of the same macroplace in a different context:

(UserGroup(PHYS) AND Site(HOSPITAL) AND AdmissionMode(OUTPATIENT) AND PatientSex(F))

(Figure 5), we note that its value is equal to $3B_i + 1L + 2S_i + 8T$ (two stack text settings and two textfields less).

We lack a theory about display complexity which parallels KLM theory about action complexity: and therefore cannot estimate, from such an expression, the cognitive load that is associated with these estimates of display complexity. Rather than evaluating a single solution, this measure supports comparative evaluation of alternatives. The designer will have the possibility of assessing which is the effect, on action complexity, of organizing a specific set of data into several, simple windows in cascade rather than in a unique, more complex, one; this will make analysis of the action-display complexity trade-off easier.

6. An outline of validation

In this paper we presented a formal method to design, simulate and evaluate context-sensitive interfaces. The method was defined and employed initially to specify an interface under Xview, but can be applied to other interaction styles, provided that interaction objects and techniques that are available in that style are described preliminary. In making some final considerations about this method, we will refer to the evaluation criteria that Davis proposes for Software Requirements Specification (SRS) techniques (Davis, 1988): (1) when the technique is properly used, the SRS should be helpful and understandable to non-computer-oriented customers and users. (2) it should be able to serve effectively as the basis for design and testing and (3) it should provide automated checks for ambiguity, incompleteness and inconsistency.

As we anticipated in Section 2, the reasons why we designed XDM were exactly those specified in Davis’ criteria and, after an informal evaluation, we claim that it responded to these needs when used as a cooperative tool in our projects. We now employ it in other research projects and also in our HCI teaching activities. Students describe, simulate and evaluate their interfaces with this method. By looking at the nets, they discover the main
errors and shortcomings in their design (deadlocks, violations of access rights and so on); by using the tool in the “Simulation” and “Verification” modes, they learn how to make more careful usability evaluations.

To make a more formal validation of our technique and tool, we should contrast it with a different formal specification technique or with some empirical evaluation of fast prototypes (for instance, cognitive walkthrough). The first comparison would be preferable, as formal methods and fast prototypes should be considered as complementary rather than alternative steps of interface design and evaluation. However, as every formal method enables representing slightly different aspects of interaction, intra-method comparisons are difficult to perform. For this reason, we are planning a validation experiment to contrast XDM with cognitive walkthrough. The experiment will include the following steps.

(a) To assess whether the method is “understandable to non-computer-oriented customers and users”, we show a model to a group of non-computer scientists, ask them to answer a list of questions about those interface properties and verify whether the answers are correct. The time taken to make this description is a measure of the technique’s helpfulness.

(b) As far as the second criterion is concerned (“the technique should be able to serve effectively as a basis for design and testing”), we measure time and errors made in designing the interface for a specified application.

(c) Assessing whether the third criterion is respected (to “provide automated checks for ambiguity, incompleteness and inconsistency”) is a more subtle issue. XDM provides these automated checks, or makes them easier to perform: however, it remains to be proved whether these checks are really easy to make for XDM users.

We verify this hypothesis by asking subjects to respond to questions concerning the interface usability (consistency, complexity and so on) first on a “correct” interface, and second on an interface in which we introduce some usability errors. The time employed to answer the questions is taken, also in this case, as a measure of how easily usability evaluations can be made with the two approaches.

All experiments will be made after defining a case study in a specific application domain, building an interface prototype for that case study and forming two groups of comparable subjects, one of which will employ XDM whereas the second one will make a cognitive walkthrough. These studies are being designed in cooperation with a group of experimental psychologists.

7. Final comments

Our work is related to a large body of research devoted to propose how to model formally the interface behaviour: for instance, State-Charts Harel, 1987). ADV Charts (Carneiro, Cowan & Lucena, 1994) or Petri net objects (Palanque, Bastide, Dourte & Sibertin-Blanc, 1993).

As we mentioned in the previous section, every formal method enables representing different aspects of interaction. The method with which XDM has more similarities, in spite of the different formalism employed, is probably GOMS. The top-down decomposition
of nested transitions in our PNs parallels the hierarchical decomposition of tasks in Kieras’ Goals; XDM’s actions correspond to GOMS’ Operators, and action sequences correspond to Methods; our transition guard functions and our logical and physical projections include knowledge represented in GOMS’ Selection rules. However, XDM models describe the display status in every phase of interaction in much more detail. In addition, due to their extended use of graphics, they show in a clear way the static and dynamic properties of the interface, also in complex situations (such as multithread and context-customized dialogues) in which other formalisms or languages are difficult to grasp. For these reasons, we claim that XDM is particularly suited to support cooperative interface design with non-computer-oriented users. Designers can employ our tool to simulate the interface behaviour in distinct contexts and to revise specification according to problems discovered. They can also check automatically whether the interface is correct (that is, complete and non-ambiguous) and whether access rights are respected, and can be assisted in evaluating interface consistency. Our combination of XDM with Card, Newell and Moran’s KLM operators performs automatically quantitative evaluations of complexity.

We must admit that building XDM models is rather time consuming when the system we want to represent is not trivial. The main difficulty is in rendering correctly visual concurrency, synchronization and selection alternatives. However, top-down specification supported by Nested PNs introduces a systematization in the design that reproduces the hierarchical task analysis and therefore reduces the risk of errors. In addition, the tool supports the model building phase by enlightening shortcomings like, for instance, a redundant place or transition or a non-represented piece of information or task. After a model has been built, the designer can use it to simulate the interface behaviour also by showing to customers or to representatives of final users how interaction will look like when they will use the system. Knowledge employed by XDM is larger than knowledge embodied in a fast prototype. During the simulation, users are informed, if required, about the tasks achieved with all interaction objects and therefore about the consequences of their actions. Simulation can be interrupted to revise the model or to evaluate usability and correctness for specific tasks or for the whole interface. In our view, this interleaving of model building, simulation and evaluation makes XDM a powerful tool for iterative design.

There are at least two opportunities to further exploit the body of knowledge that is represented in XDM models: we might use it to provide some expert assistance in generating the interface [like, for example, in UIDE (Foley, Kim, Kovacevic & Murray, 1991) of GENIUS (Janssen, Weisbecker & Ziegler, 1993), or DON (Kim & Foley, 1993)]. In addition (and this is the perspective in which we are moving at present), we might couple this modelling approach with hypermedia generation techniques, to produce automatically several types of messages: context-adapted helps, messages that show the results of usability evaluation in a more plain and understandable form, and even on-line instruction manuals.

Some of this work was carried out as part of a three-year project funded by the Advanced Informatics in Medicine initiative of the EC (contract n. 2027). We are indebted to Dianne Berry, Isabelle de Zegher and Alain Venot for the fruitful discussions about UI design and evaluation which gave us several hints on how to define and refine XDM.
References


Paper accepted for publication by Associate Editor, Prof. H. Thimbleby.